

# CUDA HW3 | Exponential Integral

Patryk Drozd  
Trinity College Dublin  
drozdp@tcd.ie

## Task 1 - Cuda Implementation

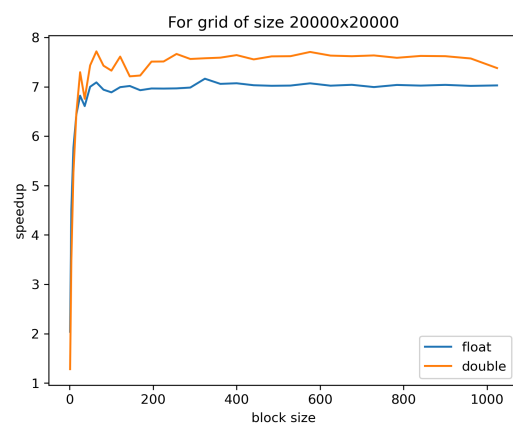
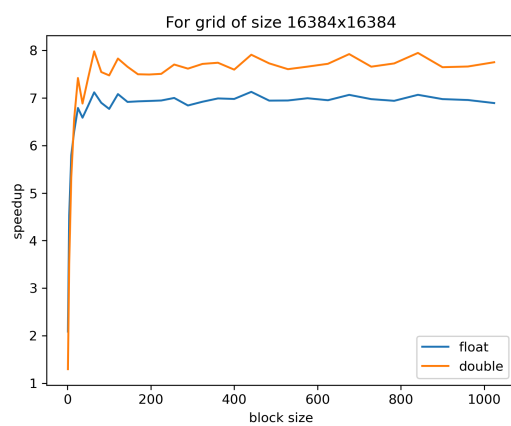
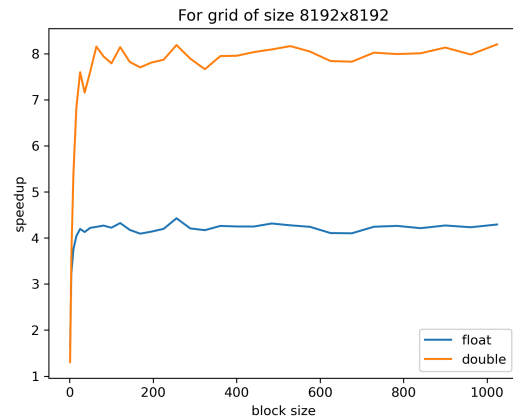
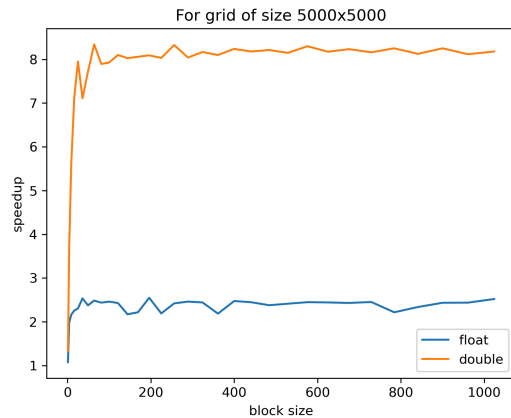
For this cuda implementation I chose to stick with using global memory and not using texture or surface memory as I did that in the last assignment. I used asynchronous memory transfers in "main.cu" to take device data and move to the host. I made a very bad dynamic parallelism function which is used which spawns a grid from one thread. I tried using constant memory but could only think of using it in the declaration of the euler constant in "funcs.h". I made few more flags to be used by the "main" executable which make writing the scripts easier for me. They are added to the "printUsage()" function but I'll paste the relevant lines here too.

```
printf("    -e          : will show the error between gpu and cpu versions of  
the code (default: no)  
");  
printf("    -B value    : will change the block size for the cuda grid (default:  
32)  
");  
printf("    -r          : will print runtime properties into a .csv style file  
(default: no)  
");
```

The following is output from "plotting.py".

```
The best block_size for floats for grid of 5000x5000: 196 = 14x14  
The best block_size for doubles for grid of 5000x5000: 64 = 8x8  
  
The best block_size for floats for grid of 8192x8192: 256 = 16x16  
The best block_size for doubles for grid of 8192x8192: 1024 = 32x32  
  
The best block_size for floats for grid of 16384x16384: 441 = 21x21  
The best block_size for doubles for grid of 16384x16384: 64 = 8x8  
  
The best block_size for floats for grid of 20000x20000: 324 = 18x18  
The best block_size for doubles for grid of 20000x20000: 64 = 8x8
```

And the performance against block size.



## Task 2 - LLM Implementation

I used chat GPT and used the prompt “explain and make a cuda parallelised version of the following code” I then copy and pasted “main.cpp” Then got output of the following

<https://chatgpt.com/share/682c696e-bc9c-8004-a268-d64d4be1cf29>

I divided the generated code into how I understood the llm layed them out in the directory “LLMexponentialIntegral” along side the text output in “llm\_output.txt”.

For the most part the llm genenated correct results which look quite similar to my implementation. In their implementaion of the cuda kerel the only provide a version for floating point precision while I implemented a templated version which can proccess floats or double. It also used a 1 dimensional grid as opposed to the 2d grid I used for indexing which I dont know if it makes a difference for this problem, other than having to think about the indexing a bit more in 1d.

In its “main.cpp” file uses a “cudaDeviceSynchronise()” function call right after the cuda kernel is called which is not neccessary since “cudaMemcpy(...)” is synchronous to the host, I found this int the cuda docs for this function

[https://docs.nvidia.com/cuda/cuda-runtime-api/group\\_\\_CUDART\\_\\_MEMORY.html#group\\_\\_CUDART\\_\\_MEMORY\\_1gc263dbe6574220cc776b45438fc351e8](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY_1gc263dbe6574220cc776b45438fc351e8)